EECS 3201 - Lab #3 Report
Addition and Subtraction Calculator

Group 04 Section B:
Ajay Deonandan #217987215
Isaiah Gocool #218918052

Overview 3
Objective 3
Acronyms 3
Design Approach 3
Inputs and Outputs 3
Displaying Numbers on Displays 3
Displaying 1, Minus or Blank on HEX1 4
Performing Arithmetic 4
Test Strategy 5
Development Testing Strategy 5
Issues Encountered 5
Final Testing Strategy 6

Overview

Objective

In lab 3, we were tasked with implementing a four-bit ripple carry adder that could also perform 2's complement subtraction in Verilog which would then be demonstrated using a DE10-Lite board. This adder would use the flip switches of the DE10-Lite board as inputs for the four-bit numbers as well as a switch to indicate operation. The board's 7-segment displays would be used to output the sum or difference in hexadecimal.

Acronyms

The 10 DE10-Lite toggle switches are referred to as SW[N] where N is the number of the switch being referred to. The 7-Segment displays are referred to as HEXN where N is the display number referred to. In this lab, switches 0 - 7 and switch 9 are utilized along with 7-segment displays 5 (display all the way to the left), 3 (display in the middle), 1 (second display to the right) and 0 (display all the way to the right).

Design Approach

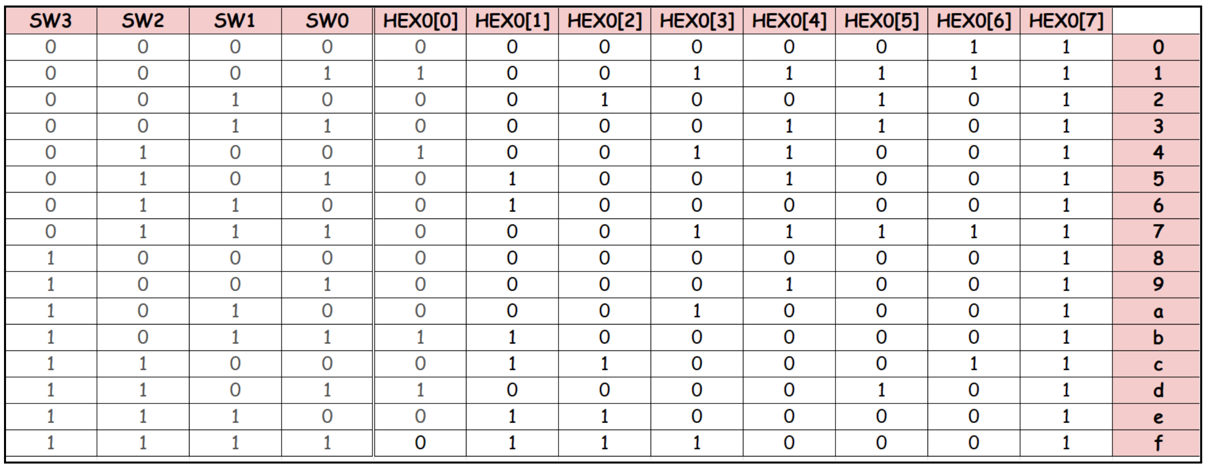
Inputs and Outputs

On the DE10 Lite board, switches 0-3 (SW[3:0]) were used to assign the first input that was displayed on the HEX3 display. Switches 4-7 (SW[7:4]) were used to assign the second input and were displayed on HEX5. Switch 9 (SW[9]) was used to determine the operation used. If it was 0, then addition would be performed and if it was 1 subtraction would be performed.

Displaying Numbers on Displays

In order to display hexadecimal numbers onto the 7-Segment Hex displays, we decided to integrate our lab 2 part B module as part of our code, allowing us to take a four-bit number and display its corresponding hexadecimal number. Throughout our code for lab 3, we had to display numbers on three various hex displays. By utilizing our completed lab 2 code, we simply had to pass in our four-bit number we wanted to display as parameters for the lab 2 module as well as the corresponding hex display we wanted this number to be displayed on.
Furthermore, we utilized part b from lab 2 as it only required two bit buses to be passed as parameters rather than 4 individual bits for the number being displayed and 7 bits from the top level file for each hex display segment which sped up the coding process as well as made our code easier to read. This allowed us to use the 4-bit numbers created with SW[3:0] and SW[7:4] to be displayed on HEX5, HEX3. It also allowed us to easily display our calculated result (the 4-bit sum) on HEX0.

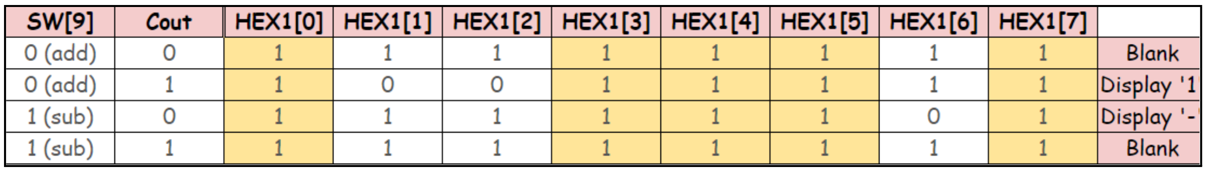
In order to turn on the correct segments for numbers 0 - F, we created a truth table showing which segments to turn on and off for each corresponding hexadecimal number depending on a 4 bit input. This truth table can be seen in figure 1.



SW3	SW2	SW1	SW0	HEX0[0]	HEX0[1]	HEX0[2]	HEX0[3]	HEX0[4]	HEX0[5]	HEX0[6]	HEX0[7]	
0	0	0	0	0	0	0	0	0	0	1	1	0
0	0	0	1	1	0	0	1	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	1	2
0	0	1	1	0	0	0	0	1	1	0	1	3
0	1	0	0	1	0	0	1	1	0	0	1	4
0	1	0	1	0	1	0	0	1	0	0	1	5
0	1	1	0	0	1	0	0	0	0	0	1	6
0	1	1	1	0	0	0	1	1	1	1	1	7
1	0	0	0	0	0	0	0	0	0	0	1	8
1	0	0	1	0	0	0	0	1	0	0	1	9
1	0	1	0	0	0	0	1	0	0	0	1	a
1	0	1	1	1	1	0	0	0	0	0	1	b
1	1	0	0	0	1	1	0	0	0	1	1	c
1	1	0	1	1	0	0	0	0	1	0	1	d
1	1	1	0	0	1	1	0	0	0	0	1	e
1	1	1	1	0	1	1	1	0	0	0	1	f

Figure 1: Truth table for the 7-segment display (0 = turn segment light off, 1 = turn segment light on)

The switches would determine the four-bit binary number input, while the HEX0 values would be assigned either a 1 or a 0 depending on what the binary number is to display a certain hexadecimal number. The combination of HEX0 values display this certain value in the last column of the truth table.



SW[9]	Cout	HEX1[0]	HEX1[1]	HEX1[2]	HEX1[3]	HEX1[4]	HEX1[5]	HEX1[6]	HEX1[7]	
0 (add)	0	1	1	1	1	1	1	1	1	Blank
0 (add)	1	1	0	0	1	1	1	1	1	Display '1
1 (sub)	0	1	1	1	1	1	1	0	1	Display '-
1 (sub)	1	1	1	1	1	1	1	1	1	Blank

Figure 2: Truth table for determining the sign of the sum/difference (0 = turn segment light off, 1 = turn segment light on)

Displaying 1, Minus or Blank on HEX1

For the truth table in figure 2, we followed the instructions given in the lab manual for determining whether HEX1 should be turned on or off based on the value of Cout and the operation used. Switch 9 was used as the input because if it was on, then the operation would be subtraction and vice versa. To fill in the rest of the values for the truth table, we used the information in the lab manual. For example, it said that for addition, if Cout was equal to 1, then 1 should be displayed on HEX1. This corresponds to the second row in our truth table, where SW[9] was set to 0 (because of addition), Cout was set to 1, and then HEX1[0:7] were assigned either a 1 or a 0 so that HEX1 would display a 1 on the board.

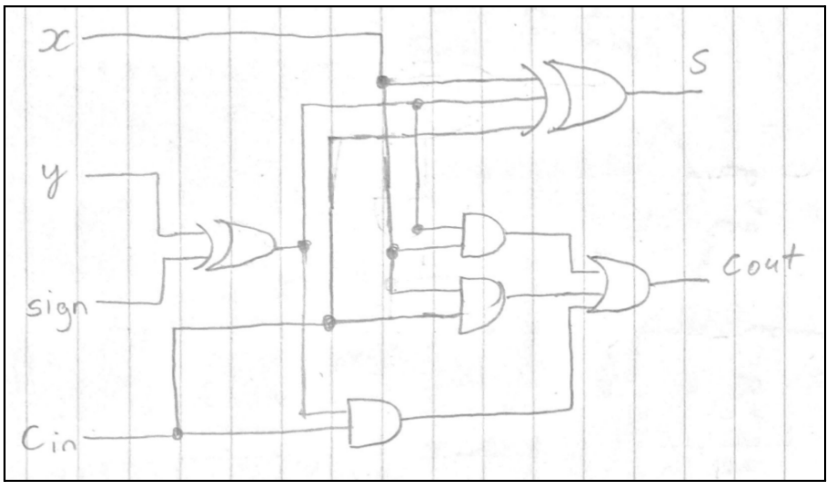


Figure 3: Circuit diagram for the full-adder.

Figure 3 shows our implementation of the full-adder circuit. The y and sign variables needed to be XORed because when the sign is 1, this indicates we are doing subtraction. When you subtract, you need to perform 1s complement by inverting all the bits. If the sign is 0, then when you XOR y and the sign, y stays the same. This output is then XORed with x and Cin to produce the sum. To calculate Cout, an OR gate was used with three inputs. In order to simplify the inputs, the result of y XOR sign will be z. The first input was x AND z, the second was x AND Cin, and the third was Cin AND z.

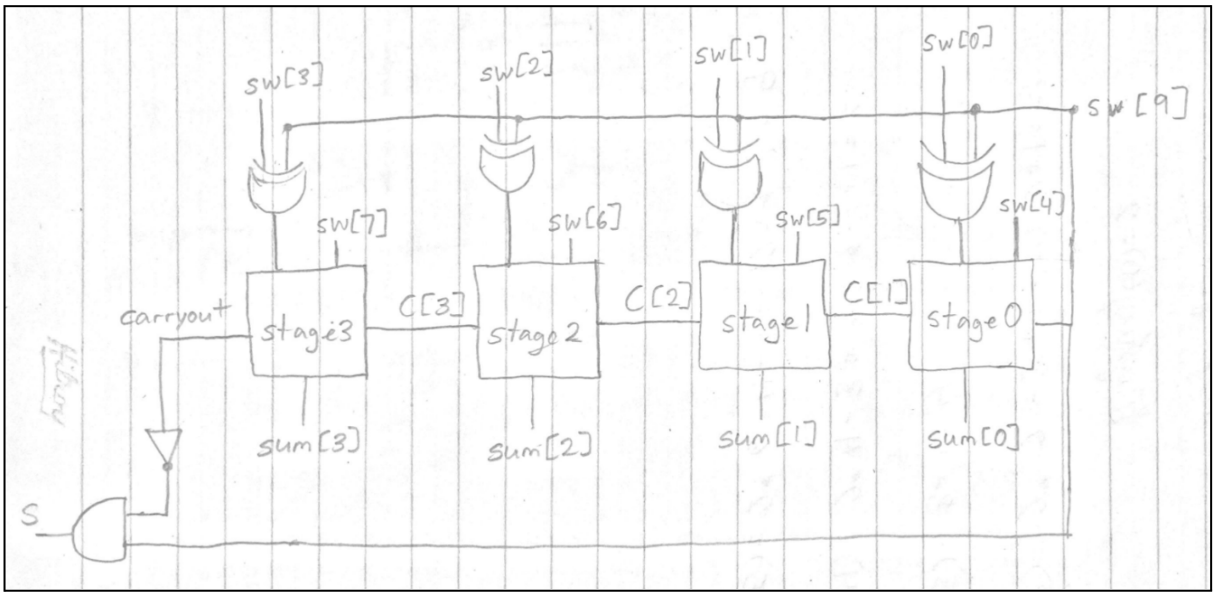


Figure 4: Circuit diagram for the four-bit adder.

In our final four-bit adder, we used our full-adder four times to calculate the sum and carry-out for each bit in the addition/subtraction. As mentioned before, switches 0 - 3 would represent the y variable, while switches 4 - 7 would represent the x variable. Each bit would be processed in the aforementioned full-adder to calculate the sum and carry-out. At the very end of our circuit, the last carryout would be inverted and ANDed with switch 9 (which was used to determine the operation used). We did this to check if our sum is in 2's complement. If it is in 2's complement form, the sum needs to be turned back into a 4-bit number by inverting each bit and then adding 1. If it isn't in 2's complement form, then the sum can be left unchanged. Afterwards, this four bit sum that had been calculated was displayed using the same method that we used to display the other 4 bit inputs into hexadecimal onto the 7-segment displays (Lab #2's module).

Test Strategy

Development Testing Strategy

To develop the code for our addition and subtraction calculator, our group split this lab into several smaller parts which allowed us to focus on one problem at a time. This allowed us to develop smaller blocks of code that could be easily tested for correctness on the board. We first started with displaying a hexadecimal number onto one of the 7-segment displays that could change depending on what switches were turned on and off (Our Lab 2 part B module was used to do this). After we verified that one display was displaying a hexadecimal number based on corresponding switches, we duplicated this code to work for another display and set of switch inputs.

We then moved onto our next task which was to create a ripple adder to add 2 4-bit numbers. While implementing this, we first manually inputted the two numbers being added together in the adder to test to see if the adder was working, then implemented it so that our 4-bit numbers created with the switches acted as the two numbers being added together. The result of this addition was displayed on HEX0 using the same strategy we used to display our input numbers (this time instead of using 4 switch input bits, we used the 4 sum bits). To test to see if our addition was working correctly, we added smaller numbers that would have a sum between 0 and F as well as used larger numbers in order to see if our sum's one place value matched that of what was displayed on HEX0.

After verifying that our adder was working correctly, we implemented the code for determining if HEX1 should contain a blank, 1 or minus symbol. We tested to see if a 1 was displayed on HEX1 by adding larger numbers that had a sum between 10 and 1F. To make sure the blank was working correctly, we used smaller numbers that would have a sum between 0 and F.

After we verified that addition was working correctly, we dealt with subtracting part of the lab. We first created another full adder that did one's complement addition and displayed it the same way we displayed the sum for our ripple adder. We commented out the parts used for ripple addition as we were now only testing subtraction. We used numbers that would give a difference between 0 and F as well as numbers that would give a difference between -F and 0.

Issues Encountered

While testing subtraction, we noticed that there would be some cases when an incorrect number would be displayed on HEX0. When comparing the answer displayed on our calculator versus the actual answer when performing the calculation on paper, we noticed that we were displaying the two's complement version of the result rather than displaying it in hexadecimal. This made us come to the realization that we needed to check to see if our result was in two's complement and revert it back to a regular 4-bit number when necessary. We used an always block to check this and made subtraction work correctly. Afterwards, we implemented a way to tell if we were subtracting when a switch (SW[9]) was turned on and to only perform subtraction when this switch was on.

Final Testing Strategy

After we had an addition and subtraction working, we then decided to do a final test of our calculator using a mixture of small and large numbers. For addition, we checked to see if our calculator worked for two numbers that would sum to numbers from 0 - F, and then numbers that would sum to anything from 10 - 1F. This allowed us to see if our overflow worked correctly as well as normal addition. For subtraction, we used large numbers subtracted with a small number that would result in anything from 0 - 9 then A - F. This allowed us to check if normal subtraction was working (without needing a minus symbol) and would also allow us to see if we were displaying the hexadecimal number instead of a two’s complement number. We then tested to get a result from -9 - 0 and then -F - (-A) using a smaller number and a large number. This allowed us to see if we were displaying negatives correctly with our minus symbol as well as displaying hexadecimal over two’s complement.